**Featherman’s T-SQL Anaytics Adventures© – More on Arrays**

This document supplements prior arrays (table variables) content which explains array theory and history. Here we further demonstrate some typical usage patterns, and demonstrate new UPDATE SET and INSERT strategies.

**Arrays Example #1** – Let’s start with a review in case you need it. As part of an INSERT SQL statement, a simple GROUP BY sub-query on lines 12-16 loads two columns of an array, and an UPDATE SET command is used to fill the other two columns.

|  |  |  |  |
| --- | --- | --- | --- |
| *This query shows a common pattern a) load some local variables with summary values b)* ***DECLARE*** *(create) an array and then use an* ***INSERT INTO SELECT FROM WHERE*** *command to load some columns of the array,  c) use* ***UPDATE SET*** *commands to calculate columns of metrics inside the array d) display the results and values of the array* | | Lines 1- 7 creates a local decimal variable and then assigns a value to it by running a SELECT aggregating query that returns one number from the fact table - the total number of units sold. Notice that this variable facilitates line 19. **It is very common to calculate and pull a value from a database and use it to calculate new columns of analytics, using this approach.** *The next module shows a short-cut for this functionality.*  The Line 6 PRINT statement produces this message, which shows the value for the @variable.  Line 8 creates the array with 4 columns.  Lines 11 – 17 adds rows to two columns of the array. An INSERT INTO (columns of an array) query is used to push the results retrieved from a SELECT FROM WHERE sub-query.  Line 19 & 20 show how to use an UPDATE SET command to fill the first column of the array, and the [% of total Units] analytics column.   ***Creating new columns of analytics is how you produce the insight to solve business problems.*** | |
| *-- Copy/paste this code to run it and play.*  USE [Featherman\_Analytics];  DECLARE @TotalBikeUnits decimal  = (SELECT SUM(rs.[OrderQuantity])  FROM [dbo].[FactResellerSales] as rs  INNER JOIN [dbo].[AW\_Products\_Flattened] as p  ON p.[ProductKey]= rs.[ProductKey]  WHERE [Category] = 'Bikes')  PRINT @TotalBikeUnits  DECLARE @BikesData TABLE ([Channel] nvarchar(15),[Model] nvarchar(25), [#Units] decimal(10), [% Total Units] nvarchar(8) )  INSERT INTO @BikesData ([Model], [#Units])  SELECT [Model], SUM(rs.[OrderQuantity])  FROM [dbo].[FactResellerSales] as rs  INNER JOIN [dbo].[AW\_Products\_Flattened] as p  ON p.[ProductKey]= rs.[ProductKey]  WHERE [Category] = 'Bikes'  GROUP BY [Model]  UPDATE @BikesData SET  [Channel] = 'Reseller'  , [% Total Units] = FORMAT(([#Units]/@TotalBikeUnits), 'P1')  SELECT \* FROM @BikesData  *Yes its worth repeating: This query shows a common pattern*  *a) load some local variables with summary values b)* ***DECLARE*** *(create) an array and use an* ***INSERT INTO SELECT FROM WHERE*** *command to load some columns of the array c) use* ***UPDATE SET*** *commands to calculate the metrics for some columns in the array d) display the results and values of the array* | Here is the code highlighted to assist your learning. To learn this content, copy/paste the code on the left and run it.  Next learn by experimenting. Change the granularity of the query, change some field names, add different array columns to experiment. INSERT new column values from a GROUP BY query into some columns.  The blue highlighted SQL code on the left references the array, notice that the array is DECLARE(d), then rows of data are INSERT(ed) INTO two columns of the array. Next two columns are filled with data using an UPDATE SET command. Finally, to display the output above, we use a SELECT command.  This is a very common SQL analytics and ETL pattern of data integration, summarization, and analytics formation (as exemplified by the [% Total Units] column. Once the data is in the array, it is easier to work with.  IMPORTANT: *Notice the array column* [% Total Units]. *It was given the datatype nvarchar so that the % sign can be added. This greatly appreciated formatting unfortunately turns the columns data into textual data which can no longer be used in calculations.*  *DBA’s will tell you that the final formatting of the data is best handled at the Tableau/Excel/PowerBI level.*  *If your query is just one step in further data processing, then leave the numeric data as numeric.* | |

|  |  |
| --- | --- |
| *While line 6 to 12 INSERT many rows into the array,  line 19-20 INSERTS one row* | **Arrays Example #2 - Extensions**  This query improves and extends the first example **in four ways.**  **1**. Notice line 14. It is totaling one of the columns of the array. The [#Units] column of the array is loaded using the code in lines 6 to 12. Line replaces lines 2-6 in the query above. How is it possible to replace 5 lines of code with one? Answer: Because the data that needs to be totaled is already inside your array.  The first example queries the database two times, once at a granularity of Model and once at the granularity of ALL Models.  Because the data comes from the same tables, you can bring the data into the array and then total a column from the array. This is what line 14 accomplishes.  If you need to pull data from different databases then use lines 2 -6 in the first example. Analysts often need to pull data from different databases into one dataset. Adding additional datasets can help to further understand the phenomenon.  **2**. Line 19 and 20 insert a row into the array to show summary data that was calculated. |

|  |  |
| --- | --- |
|  | Here is the same code but without use of a local variable. Compare the first red box with line 16 & 17 above. While it is easier to read the code where you load a value into a variable and then use that variable in other calculations, you can use a different approach.  Notice line 15 contains a sub-query which requires the use of parentheses.  On line 15 or 18, if the line of code gets too complex then by all means use a local variable as in the prior approach.  UPDATE SET – if you have a set of rows, then the UPDATE SET fills one or more column with values. UPDATE SET does not add more rows, rather it updates rows that pre-exist.  INSERT INTO adds more rows to a table |

|  |  |
| --- | --- |
| **Arrays Example #3**      *Copy/paste this code into SSMS and learn*  USE [Featherman\_Analytics]; -- create the array  DECLARE @Summary TABLE([Channel] nvarchar(10), [# SubCat] decimal(4), [# Units] decimal(10)  , [% Total] decimal(8,2), [Profit] decimal(10,2), [% Profit] decimal(8,2))  DECLARE @TotalUnits decimal(8),@Profit decimal(8),@AvgSubCat decimal(8)  *-- retrieve data from one table into the array - create one compiled row of data*  INSERT INTO @Summary ([Channel], [# SubCat], [# Units], [Profit])  (SELECT 'Reseller', COUNT(DISTINCT(p.[Sub Category])), sum(rs.[OrderQuantity])  , SUM(CAST([SalesAmount] -[TotalProductCost] AS decimal))  FROM [dbo].[FactResellerSales] as rs  INNER JOIN [dbo].[AW\_Products\_Flattened] as p ON p.ProductKey = rs.ProductKey)  *-- create a second row of data to display data from a second data source*  INSERT INTO @Summary ([Channel], [# SubCat], [# Units], [Profit])  (SELECT 'Web', COUNT(DISTINCT(p.[Sub Category])), sum(i.[OrderQuantity])  , SUM(CAST([SalesAmount] -[TotalProductCost] AS decimal))  FROM [dbo].[FactInternetSales] as i  INNER JOIN [dbo].[AW\_Products\_Flattened] as p ON p.ProductKey = i.ProductKey)  *-- calculate and store values into three local variables*  SET @TotalUnits = (SELECT SUM([# Units]) FROM @Summary)  SET @Profit = (SELECT SUM([Profit]) FROM @Summary)  SET @AvgSubCat = (SELECT AVG([# SubCat]) FROM @Summary)  *-- Calculate the values for two columns of analytics*  UPDATE @Summary SET [% Total] = ([# Units]/@TotalUnits), [% Profit] = ([Profit]/@Profit)  *-- Add a summary row to the datatable to display the calculated totals*  INSERT INTO @Summary VALUES ('Total', @AvgSubCat, @TotalUnits, 1, @Profit, 1)  SELECT \*FROM @Summary  *-- Display nicely formatted data - leaving the underlying data in numeric columns*  SELECT [Channel], [# SubCat], FORMAT([# Units], 'N0') as [Total #Units]  , FORMAT([% Total], 'P1') as [% Total Units], FORMAT([Profit], 'N0') as [Total Profit]  , FORMAT([% Profit], 'P1') as [% Total Profit]  FROM @Summary | This example uses a similar pattern CREATE the array, INSERT data into 4 columns, calculating some summary totals from the data in the array, creating a few columns of analytics with an UPDATE SET command (adding profit analysis). We again add a row of summary data and display the data. Here the final presentation of the array data adds the formatting. The data in the underlying array is not reformatted into textual content to provide the currency or % formatting.  *New features of this query*  1. Lines 7 – 11 create a row of data to summarize the ResellerSales, and lines 14-18 summarize the Web sales. The SUM and COUNT functions tell you that the data is being grouped by whatever dimension attributes are in the SELECT statement. Here four columns are retrieved/aggregated and INSERTED INTO the array using the INSERT INTO SELECT FROM WHERE pattern. The queries each return one row so no GROUP BY is used.  The CAST() as decimal statement was necessary as the columns were not the right datatype.  2. The query has three INSERT INTO code blocks. In this case each INSERT statement adds one row of heavily summarized data.  3. Lines 21-23 again use local variables to create summary measures that are reported on line 29, and also used to produce other columns of analytics (line 26, 29)  4. This example provides the extension of a profitability analysis.  This is an example of transforming data from fact tables and adding rows of summary data for further analysis. Use code like this to compile data at different levels of summarization. Once the data is in the array you can aggregate or average data in columns, to allow further analysis (here % of total). You can add new columns of metrics (UPDATE SET) , new rows of data (INSERT), and new calculated values stored into local variables and further used.  This array demonstrates different ways to shape data (new rows, new columns new variables). Each of these concepts are used to seek more and more insight. The goal is to put data together to investigate business performance. A well formed dataset can provide insight to managers.  Recommendation: Use decimal datatypes for numeric columns,  In the final display, FORMAT() if you want to clean up the display. |

|  |  |
| --- | --- |
| USE [Featherman\_Analytics];  BEGIN  DECLARE @TotalBikeUnits decimal, @TotalProfitLoss decimal  , @TotalBikeProfit decimal, @TotalBikeLosses decimal, @AvgProfitPerUnit decimal  --Declare local variables (above) and the array (below)  DECLARE @BikesData TABLE ( [Channel] nvarchar(15), [Model] nvarchar(25)  , [#Units] decimal(10), [Profit] decimal(10), [% Total Units] decimal(8,2)  , [Profit/Loss Per Unit] decimal(8), [% of Total Profit] decimal(8,2)  , [% of Generated Profits] decimal(8,2), [% of Generated Losses] decimal(8,2) )  END  -- Load values into three columns of the array  INSERT INTO @BikesData ([Model], [#Units], [Profit])  SELECT [Model], SUM(rs.[OrderQuantity]), SUM([SalesAmount]-[TotalProductCost])  FROM [dbo].[FactResellerSales] as rs  INNER JOIN [dbo].[AW\_Products\_Flattened] as p ON p.[ProductKey]= rs.[ProductKey]  WHERE [Category] = 'Bikes'  GROUP BY [Model]  ORDER BY [Model]  -- Load values into local variables  SET @TotalBikeUnits = (SELECT SUM([#Units]) FROM @BikesData)  SET @TotalProfitLoss = (SELECT SUM([Profit]) FROM @BikesData)  PRINT @TotalProfitLoss  -- Update columns in the array  UPDATE @BikesData SET [Channel] = 'Reseller'  , [% Total Units] = ([#Units]/@TotalBikeUnits)  , [Profit/Loss Per Unit] = ([Profit]/[#Units])  , [% of Total Profit] = ([Profit]/@TotalProfitLoss)    SELECT \* FROM @BikesData | Run this query  Oh No this query doesn’t work - % of profit calculations get FUBAR when the overall number is a loss.  Can a model that is making money be a % of losses? |

|  |  |  |  |
| --- | --- | --- | --- |
| **Arrays Example 4** | This last example shows some useful extensions, mostly on the next page.  The program again starts creating local variables on line 2 and 3 which will be needed for calculations. Lines 6 – 9 create the array, and define the column names and their data types.  Lines 12-18 load data into three columns of the array. The data provides totals for units sold and profit grouped by Model.  Lines 21 – 26 calculate column totals and assign the calculated values to local variables.  Due to problems calculated % of total profit, lines 25, and 26 will be used to display the percent of profit generated by models (when they do generate a profit), and % of loss (when the models generate a loss).  The problem is that the bikes category is losing almost a million dollars. So any calculation of % of total profit is not possible to calulate. This example produces new metrics which provide insight into which models are gnerating profits, and which models are generting losses. To support these calculated columns, the total Bike profits and total bike losses need to be calculated (line 25, 26).  Lines 29-32 perform an UPDATE SET command that fill three columns of the array. We will see that the line 31 calculation provides unusual results, check the [% of Total Profit] column in the output below. | | |
|  | | Lines 34 – 43 show two examples of UPDATE SET commands that needed a CASE structure inside them. The CASE syntax used here provides the needed results, however is relatively simplistic; more levels of WHEN THEN processing could be utilized.  You may think that the formatting used is an unnecessary use of space, making the query too long. Many analysts and DBA’s use the formatting to reduce errors and make their code easier to read.  Lines 34-37 place a calculated value in the column only for rows where the value in the [Profit] column are more than zero.  When the value in the [Profit] column is not > 0 then a 0 is placed in the column. A value is placed in the column for every row.  Lines 39-42 are similar, but only place a calculated value in the column when the value in the [Profit] column are negative (less than zero).  Line 46 demonstrates that an INSERT INTO statement can place values into some not all of the columns of the array. If the columns to be loaded with values are not specified in the parentheses, then a value must be supplied for all the columns.  Line 51-52 provides an example of an UPDATE SET statement with a WHERE condition. Here we ensure only 1 row of data is updated. | |
|  | | You may also want to save the array contents and structure into a database table, so that a report writing software can connect to the table.  This is demonstrated on lines 67-68, which create the SQL database table with the same structure (compare the image on the left to the array structure in the DECLARE statement).  Line 66 deletes the table if it previously exists, allowing the table to be created and filled with data again. This is a fast but a little heavy-handed.  Another prodedure is to rather delete the rows of an existing table and refill it with the new data from your array. In essence running the query refreshed the data and all the charts and metrics.  Line 67 created the BikesData table shown to the left in the database. | |
| USE [Featherman\_Analytics];  BEGIN  DECLARE @TotalBikeUnits decimal, @TotalProfitLoss decimal  , @TotalBikeProfit decimal, @TotalBikeLosses decimal, @AvgProfitPerUnit decimal  --Declare local variables (above) and the array (below)  DECLARE @BikesData TABLE ( [Channel] nvarchar(15), [Model] nvarchar(25)  , [#Units] decimal(10), [Profit] decimal(10), [% Total Units] decimal(8,2)  , [Profit/Loss Per Unit] decimal(8), [% of Total Profit] decimal(8,2)  , [% of Generated Profits] decimal(8,2), [% of Generated Losses] decimal(8,2) )  END  -- Load values into three columns of the array  INSERT INTO @BikesData ([Model], [#Units], [Profit])  SELECT [Model], SUM(rs.[OrderQuantity]), SUM([SalesAmount]-[TotalProductCost])  FROM [dbo].[FactResellerSales] as rs  INNER JOIN [dbo].[AW\_Products\_Flattened] as p ON p.[ProductKey]= rs.[ProductKey]  WHERE [Category] = 'Bikes'  GROUP BY [Model]  ORDER BY [Model]  -- Load values into local variables  SET @TotalBikeUnits = (SELECT SUM([#Units]) FROM @BikesData)  SET @TotalProfitLoss = (SELECT SUM([Profit]) FROM @BikesData)  -- These two variables needed to make calculations work  SET @TotalBikeProfit = (SELECT SUM([Profit]) FROM @BikesData WHERE [Profit] > 0)  SET @TotalBikeLosses = (SELECT SUM([Profit]) FROM @BikesData WHERE [Profit] < 0)  -- Update columns in the array  UPDATE @BikesData SET [Channel] = 'Reseller'  , [% Total Units] = ([#Units]/@TotalBikeUnits)  , [Profit/Loss Per Unit] = ([Profit]/[#Units])  , [% of Total Profit] = ([Profit]/@TotalProfitLoss)  UPDATE @BikesData SET [% of Generated Profits] =  ( CASE WHEN [Profit] > 0 THEN ([Profit]/@TotalBikeProfit)  ELSE 0  END )    UPDATE @BikesData SET [% of Generated Losses] =  ( CASE WHEN [Profit] < 0 THEN ([Profit]/@TotalBikeLosses)  ELSE 0  END )  -- This next line of code demonstrates that you can insert a new row with only a portion of the columns  INSERT INTO @BikesData ([Channel], [Model], [#Units])  VALUES ('Reseller', 'All Models',@TotalBikeUnits)  -- Calculate a summary value and store it into one row of the array  SET @AvgProfitPerUnit = (SELECT AVG([Profit/Loss Per Unit]) FROM @BikesData)  UPDATE @BikesData SET [Profit/Loss Per Unit] = @AvgProfitPerUnit  , [Profit] = @TotalProfitLoss  WHERE [Model] = 'All Models'  SELECT \* FROM @BikesData  --Format and display the data  SELECT [Channel], [Model], FORMAT([#Units], 'N0') as [# Units]  , FORMAT([% Total Units], 'p0') as [% Total Units]  , FORMAT([Profit], 'N0') as [Profit/Loss]  , FORMAT([Profit/Loss Per Unit], 'N0') as [Profit/Loss Per Unit]  , FORMAT([% of Total Profit], 'p0') as [% of Total Profit- Don't Use]  , FORMAT([% of Generated Profits], 'p0') as [% of Generated Profits]  , FORMAT([% of Generated Losses], 'p0') as [% of Generated Losses]  FROM @BikesData  -- Create a new SQL Server table and then save the data from the array into the database table. If the table already exists, then use line 50 to delete the table before recreating it.  DROP TABLE [Featherman\_Analytics].[dbo].[BikesData]  SELECT \* INTO [Featherman\_Analytics].[dbo].[BikesData]  FROM @BikesData  -- Display totals for comparison  PRINT 'Original Calculation Total Profit/Loss: ' + FORMAT(@TotalProfitLoss, 'N0')  PRINT 'Total Profit: ' + FORMAT(@TotalBikeProfit, 'N0')  PRINT 'Total Loss: ' + FORMAT(@TotalBikeLosses, 'N0')  PRINT 'Net profitability: ' + FORMAT(@TotalBikeProfit + @TotalBikeLosses, 'N0') | | | Here is the code – please watch the video and run this code in SSMS. Make changes to learn how the code works |

From Featherman

The main message of this document is that the analyst/DBA can use SQL to

a) use INSERT statements pull data from different datasources into an array

b) use local variables to aid production of columns of analytics  
  
c) use UPDATE SET commands to fill columns for the rows you already have in the array. UPDATE SET commands are etremely useful to recalculate data and produce columns of analytics.  
  
d) once data is in arrays the columns can be compared, totaled, averaged and many other analytics produced. Just use your imagination and business instinct.  
e) you can add more columns to your array from a new datasource, as long as the new data is categorized by the same dimension (ie state or model, etc.)

After you produce an amazing dataset, you can save the query and run it anytime needed. You can also place the query inside a report to automate the reporting refresh process. The next module covers these organizing tasks.

Well there is enough here to close out this document. No need to get overwhelmed, focus on the usage (shown here) of the INSERT INTO and UPDATE SET commands. We will use them repeatedly. Arrays really do make producing analytics easier. You are freed from writing 6 table JOINS that don’t seem to work.

Happy coding,

Mauricio